Computer Architecture - HW 3

Connor Finley

September 30, 2017

1

Implement the following C code in MIPS assembly. Show the contents of the stack after the function call to the function `compare` is made. Assume that the stack pointer is originally at address `0x7ffffffc`.

```
int compare(int a, int b) {
    if (sub(a, b) >= 0) return 1;
    else return 0;
}
int sub(int a, int b) {
    return a - b;
}
```

```
compare:
    addi  $sp, $sp, -4        # decrement $sp to allocate stack frame
    sw    $ra, 0($sp)         # save callee registers in return address
    jal   sub                 # substract
    addiu $t0, $zero, $zero   # init result to 0
    bltz  $v0, exit           # exit if sub(a, b) < 0 (inverse of initial inequality)
    addiu $t0, $zero, 1       # else, result = 1
exit:
    move  $v0, $t0            # $v0 = result
    lw    $ra, 0($sp)         # restore return address
    addi  $sp, $sp, 4         # free the memory of the stack frame
    jr    $ra                 # return
sub:
    sub  $v0, $a0, $a1        # $v0 = a - b
    jr   $ra                  # return
```

`$sp` returns back to `0x7ffffffc`, `$v0` returns 0 if a - b < 0, 1 otherwise

2

Implement the following C code in the table in MIPS assembly. Show the contents of the stack after the function call to the function `fib_iter` is made. Assume that the stack pointer is originally at address `0x7ffffffc`.

```
int fib_iter(int a, int b, int n) {
    if (n == 0) return b;
    else return fib_iter(a+b, a, n-1);
}
```

```
fib_iter:
    bne   $a2, $0, else      # else if n not 0
    move  $v0, $a1           # init result = b
    jr    $ra                # return
else:
```

```
    addiu $sp, $sp, -4        # decrement to allocate stack frame
    sw    $ra, 0($sp)         # store return addr
    addu  $t0, $zero, $a0     # $t0 = $a0
    addu  $a0, $a0, $a1       # $a0 = a + b
    addu  $a1, $zero, $t0     # $a1 = a
    addiu $a2, $a2, -1        # $a2 = n-1
    jal   fib_iter            # recursive call
    lw    $ra, 0($sp)         # bring back return addr
    addiu $sp, $sp, 4         # free stack frame memory
    jr    $ra                 # return
```

- Each time `fib_iter` is called, the stack pointer allocates four bytes, decrementing the address stored in `$sp`.
- `$v0` will contain the result of applying the fibonacci sequencing algorithm to a given starting point, `a` and `b` up to the `n`th iteration.

3

The following problems refer to a function `f` that calls another function `func`. The function declaration for `func` is `int func(int a, int b);`. The code for function `f` is as follows:

```
int f(int a, int b, int c) {
    return func(func(a, b), c);
}
```

3a

Translate function `f` into MIPS assembly code, using the MIPS calling convention. If you need to use register `$t0` through `$t7`, use the lower-numbered registers first.

```
f:
    addiu $sp, $sp, -8       # allocate stack frame with 8 bytes
    sw    $ra, 0($sp)        # store return addr
    sw    $a2, 4($sp)        # store c
    jal   func               # func(a, b)
    addu  $a0, $zero, $v0    # $a0 = func(a, b)
    lw    $a1, 4($sp)        # $a1 = c
    jal   func               # func(func(a, b), c)
    lw    $ra, 0($sp)        # bring back return addr
    addiu $sp, $sp, 8        # free stack frame memory
    jr    $ra                # return
```

3b

Right before your function `f` of Problem 4 returns, what do you know about contents of registers `$ra`, and `$sp`? Keep in mind that we know what the entire function `f` looks like, but for function `func` we only know its declaration.

- `$ra` equals the return address in the caller function, `$sp` has the same value it had when function `f` was called.

4

The following problems refer to a function `f` that calls another function `func`. The function declaration for `func` is `int func(int a, int b);`. The code for function `f` is as follows:

```
int f(int a, int b, int c) {
    return func(a, b) + func(b, c);
}
```

4a

Translate function `f` into MIPS assembly code, using the MIPS calling convention. If you need to use register `$t0` through `$t7`, use the lower-numbered registers first.

```
f:
    addi  $sp, $sp, -12     # allocate stack frame with 12 bytes
    sw    $ra, 0($sp)       # save return addr
    sw    $a1, 4($sp)       # store b
    sw    $a2, 8($sp)       # store c
    jal   func              # func(a,b)
    lw    $a0, 4($sp)       # $a0 = b
    lw    $a1, 8($sp)       # $a1 = c
    sw    $v0, 4($sp)       # store func(a,b)
    jal   func              # func(b,c)
    lw    $t0, 4($sp)       # $t0 = func(a,b)
    addu  $v0, $t0, $v0     # $v0 = func(a, b) + func(b, c)
    lw    $ra, 0($sp)       # bring back return addr
    addi  $sp, $sp, 12      # free stack frame memory
    jr    $ra               # return
```

4b

Right before your function `f` of Problem 4 returns, what do you know about contents of registers `$ra`, and `$sp`? Keep in mind that we know what the entire function `f` looks like, but for function func we only know its declaration

- `$ra` holds the return address of the function that was called
- `$sp` holds the value it had at the moment `f` was called

5

Write a program in MIPS assembly language to convert a positive integer decimal string to an integer. Your program should expect register `$a0` to hold the address of a null-terminated string containing some combination of the digits 0 though 9. Your program should compute the integer value equivalent to this string of digits, then place the number in register `$v0`. If a nondigit character appears anywhere in the string, your program should stop with the value -1 in register `$v0`.

```
convert:
    li    $t2, 0x30         # $t6 = '0'
    li    $t3, 0x39         # $t7 = '9'
    li    $v0, 0            # $v0 = 0
    addu  $t0, $zero, $a0   # $t0 points to string
```

```
    lb    $t1, ($t0)          # $t1 = char
L1:
    blt   $t1, $t2, nondigit # char < '0'
    bgt   $t1, $t3, nondigit # char > '9'
    subu  $t1, $t1, $t2      # convert char to integer
    mul   $v0, $v0, 10
    add   $v0, $v0, $t1      # $v0 = $v0 * 10 + digit
    addiu $t0, $t0, 1        # point to next char
    lb    $t1, ($t0)         # $t1 = next char
    bne   $t1, $0, L1        # loop if not at end of string
    jr    $ra                # return integer
nondigit:
    li    $v0, -1            # nondigit found, $v0 = -1
    jr    $ra
```

6

Repeat problem 5 with convert a string of hexadecimal digits to an integer.

```
convert:
    li    $t4, 0x41          # $t4 = 'A'
    li    $t5, 0x46          # $t7 = 'F'
    li    $t6, 0x30          # $t6 = '0'
    li    $t7, 0x39          # $t7 = '9'
    li    $v0, 0             # $v0 = 0
    addu  $t0, $zero, $a0    # $t0 points to string
    lb    $t1, ($t0)         # $t1 = char
L1:
    blt   $t1, $t6, nondigit # char < '0'
    bgt   $t1, $t7, hex      # check if hex digit
    subu  $t1, $t1, $t6      # convert to int
    j     Compute            # jump to Compute integer
hex:
    blt   $t1, $t4, nondigit # char < 'A'
    bgt   $t1, $t5, nondigit # char > 'F'
    addi  $t1, $t1, -55      # convert
    sll   $v0, $v0, 4
    add   $v0, $v0, $t1      # $v0 = $v0 * 16 + digit
    addiu $t0, $t0, 1        # point to next char
    lb    $t1, ($t0)         # $t1 = next digit
    bne   $t1, $0, L1        # loop if not at end of string
    jr    $ra                # return integer
nondigit:
    li    $v0, -1            # nondigit found, $v0 = -1
    jr    $ra
```